



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/591,986	06/12/2000	Vivek Singhal	035574-0011	5077

7590 04/23/2004

ROBERT E. KREBS
THELEN REID & PRIEST LLP
P.O BOX 640640
SAN JOSE,, CA 95164-0640

EXAMINER

BLACKWELL, JAMES H

ART UNIT	PAPER NUMBER
----------	--------------

2176

DATE MAILED: 04/23/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary

Application No.

09/591,986

Applicant(s)

SINGHAL ET AL.

Examiner

James H Blackwell

Art Unit

2176

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 12 June 2000.
- 2a) ☐ This action is FINAL. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1-29 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1-29 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 12 June 2000 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. §§ 119 and 120

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
 - ☐ Certified copies of the priority documents have been received in Application No. _____.
 - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- * See the attached detailed Office action for a list of the certified copies not received.
- 13) ☒ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application) since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.
- a) ☐ The translation of the foreign language provisional application has been received.
- 14) ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121 since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☒ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____
- 4) ☐ Interview Summary (PTO-413) Paper No(s). _____
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: _____

DETAILED ACTION

Claim Rejections - 35 USC § 102

1. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1 and 2 are rejected under 35 U.S.C. 102(e) as being anticipated by Schloss et al. (hereinafter Schloss, U.S. Patent No. 6,249,844).

In regard to independent Claim 1, Schloss teaches a method for dynamically parsing a digital content description of a named digital object, creating and maintaining fragment identities to facilitate caching (Col. 2, lines 36-39) that can be executed on a computing node (a Web server or Proxy server) (Col. 3, lines 61-67; compare to Claim 1, ***“A method of storing data pages at a proxy, the method comprising ...”***).

Schloss also teaches in Fig. 7 an object request handler. In step (705), it is first checked whether the requested object is cached in the object cache maintained by the computing node (Col. 6, lines 31-34; compare to Claim 1, ***“... (a) receiving a data page”***). Schloss also teaches a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a plurality of entries (507), where each table entry (507) points to a fragment description list (510) (only one shown for ease of description). The list (510) includes one or more

description elements (520 and 525). Each fragment that maps to a given entry in the fragment description table (510) has a unique description element (520) on the fragment description list (510) of the entry. The description element includes several fields: Nlink (530); Fname (535); and Fdescription (540). The Fname (535) is the persistent name of the fragment. This name is given by the persistent name creator routine (with details depicted in Fig. 10). The Fdescription (540) is the fragment description. The Nlink (530) points to the next description element (525) that maps to the same fragment description table entry (507) (Col. 5, lines 66-67; Col. 6, lines 1-15; compare to Claim 1, “... **(b) receiving page dependency data that contains one or more dependencies such that each dependency indicates an underlying data source which the said data page is dependent on**”). Schloss also teaches that in step (720), the computing node waits for the object requested. In step (725), after receiving the object, the object parser (with details described with reference to Fig. 8) is invoked to analyze the object description and create fragments. In step (730), the object description, which may have been modified by the object parser, is sent back to the requester. In step (735), the object cache manager is invoked to determine whether the object description (which may have been modified by the object parser) should be cached in the object cache (Col. 6, lines 37-46; compare with Claim 1, “... **(c) storing said data page; (d) storing said page dependency data**”).

In regard to dependent Claim 2, Schloss teaches an XML-like document will be used as an example of a document described using some formal language, such as a markup language. Fig. 3 shows an example of an XML-like document (Col. 4, lines 21-

24; compare to Claim 2, "... **said page dependency data are written in HTML or XML**").

Claim Rejections - 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

Claims 3-7, and 10 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schloss in view of Batchelder et al. (hereinafter Batchelder, U.S. Patent No. 6,351,767).

In regard to dependent Claim 3, Schloss fails to specifically teach that *said page dependency data are generated by a Request-based dependency generator*. However, Batchelder teaches a URL Parser (303), which breaks the URL into different parts. The parsed URL is passed to the cache control unit (311) (Col. 7, lines 13-15; compare to Claim 3, "... **said page dependency data are generated by a Request-based dependency generator**"). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Batchelder providing the benefit of having determined the cacheability of a given dynamic web page.

In regard to dependent Claim 4, Schloss fails to specifically teach *said Request-Based dependency generator uses a URL request of the said data page*. However,

Batchelder teaches a URL Parser (303) that breaks the URL into different parts. Specifically, commands to perform functions are added to the URL (after a question mark separator) (Col. 7, lines 12-27; compare to Claim 4, “... ***said Request-Based dependency generator uses a URL request of the said data page***”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Batchelder providing the benefit of having determined the cacheability of a given dynamic web page.

In regard to dependent Claim 5, Schloss fails to specifically teach that *said Request-Based dependency generator uses a configuration file to generate said page dependency data*. However, Batchelder teaches that Fig. 5 depicts an example of a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a plurality of entries (507), where each table entry (507) points to a fragment description list (510) (only one shown for ease of description). The list (510) includes one or more description elements (520 and 525) (Col. 5, lines 66-67; Col. 6, lines 1-5; compare to Claim 5, “... ***said Request-Based dependency generator uses a configuration file to generate said page dependency data***”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Batchelder providing the benefit of tracking object fragment parameters.

In regard to dependent Claim 6, Schloss fails to specifically teach that *said page dependency data are generated by a script-based dependency generator*. However, Batchelder teaches that once the web page response is built by the response builder

(307), it is passed to the HTML unit (305) for conversion to HTML. This HTML response is then passed to the HTTP server (206) for serving to the requesting user (Col. 7, lines 59-62; compare to Claim 6, “... ***said page dependency data are generated by a script-based dependency generator***”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Batchelder providing the benefit of having executed commands within software instead of typing them as part of a URL.

In regard to dependent Claim 7, Schloss fails to teach that *said page dependency data are encoded in the said data page by a script-based dependency generator*. However, Batchelder teaches that a response builder (307) uses the parsed URL to build the response by accessing the appropriate sources (via source access unit (308)) and retrieving the appropriate “parts” to construct the response. The parts retrieved by the response builder (307) may comprise many different types, including data, forms, subforms, database design elements, calculations, etc. In other words, there is no theoretical restriction as to the type of parts comprising a web page response. These parts each have their own attributes. For instance, some parts may or may not have last modified dates associated with the part. The attributes of all of the parts used to build the response are collected and analyzed by attribute analyzer (313). The attribute analyzer (313) builds a “composite” of the attributes, the attribute composite being representative of the entire response (Col. 7, lines 43-58; compare to Claim 7, “... ***said page dependency data are encoded in the said data page by a script-based dependency generator***”). It would have been obvious to one of ordinary

skill in the art at the time of invention to combine the teachings of Schloss and Batchelder providing the benefit of having had server commands and page parameters together in one file.

In regard to dependent Claim 10, Schloss fails to teach that *said storing of said data page at the proxy is in response to data in a configuration file*. However, Batchelder teaches a cacheability analyzer (309) examines the attribute composite and, if it determines that the response cannot be cached, the response is not cached. If it determines that the response can be cached, it provides an indication to the cache control unit (311), along with the response and an associated set of cache strategy indicators generated by the cacheability analyzer (309) (Col. 7, lines 64-67; Col. 8, lines 1-5; compare to Claim 10, “... ***said storing of said data page at the proxy is in response to data in a configuration file***”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Batchelder providing the benefit of enabling the determination of cacheability.

Claims 8-9 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schloss.

In regard to dependent Claim 8, Schloss fails to teach that *said page dependency data are manually encoded into a data file*. However, it would have been obvious to one of ordinary skill in the art at the time of invention to realize that one would have been able to edit a web file and manually enter the dependencies in several

ways such as placing them in the header. This would have provided the benefit of less processing on the file as dependencies would not have needed to be computed.

In regard to dependent Claim 9, Schloss fails to specifically teach *that said data page and said data page dependency data are stored in one or more files*. However, it would have been obvious to one of ordinary skill in the art at the time of invention to write code such that the data page and its dependencies would have been saved as either one or more separate files. Having a single file combining both the data page and the data page dependencies would have provided the benefit of ensuring that the dependencies would not be overridden by a separate set of dependencies. Having two separate files would have provided the benefit of using one set of dependencies for multiple data pages.

Claims 11-13, 21-23, and 27-29 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schloss in view of Arlitt et al. (hereinafter Arlitt, U.S. Patent No. 6,272,598).

In regard to dependent Claim 11, Schloss fails to specifically teach *(a) receiving an event; (b) determining if said event changes one of the said data page dependency data associated with said data page; and (c) updating the cache by refreshing or deleting said data page*. However, Arlitt teaches that the cache manager (74) controls the eviction of cached objects in the cache (72). Each storage area of the cache (72) is refreshed (i.e., evicting cached objects stored in the cache) in accordance with one of a number of replacement strategies or policies (Col. 5, lines 35-39; compare to Claim 11,

“... (a) receiving an event; (b) determining if said event changes one of the said page dependency data associated with said data page; and (c) updating the cache by refreshing or deleting said data page”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Arlitt providing the benefit of having kept the cache loaded with the most current information.

In regard to dependent Claim 12, neither Schloss nor Arlitt teach that *said event is received incorporated in an event message*. However, it would have been obvious to one of ordinary skill in the art at the time of invention to flag an event in the form of a message, in view of Schloss's and Arlitt's inventions, providing the benefit of having offered information of an event to an end user, a proxy or web manager, or a system log.

In regard to dependent Claim 13, Schloss teaches an XML-like document will be used as an example of a document described using some formal language, such as a markup language. Fig. 3 shows an example of an XML-like document (Col. 4, lines 21-24; compare to Claim 13, **“... said event is written in HTML or XML”**).

In regard to dependent Claim 21, Schloss teaches a Least-Recently-Used (LRU) algorithm (Col. 8, lines 36-38; compare to Claim 21, **“... said event came from a trigger-based event generator”**).

In regard to dependent Claim 22, neither Schloss nor Arlitt specifically teach that *said event came from a polling event generator*. However, it would have been obvious to one of ordinary skill in the art at the time of invention to use such an event generator,

in light of Schloss's and Arlitt's inventions, providing the benefit of having had a way to periodically scan for changes to a database and then sending out an alert signal when a change had occurred.

In regard to dependent Claim 23, Schloss fails to specifically teach that *said event came from a custom event generator*. However, Arlitt teaches that other known cache replacement policies may also be used for the cache (72) (Col. 7, lines 28-29; compare to Claim 23, "... ***said event came from a custom event generator***"). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Arlitt providing the benefit of having had other means to alert for changes that were unique.

In regard to dependent Claim 27, Schloss teaches a fragment cache manager that uses an Least-Recently-Used (LRU) type replacement policy (Col. 8, lines 36-38; compare to Claim 27, "... ***said updating the cache involves keeping the index of URL addresses and page dependency data up-to-date***").

In regard to independent Claim 28, Schloss teaches a method for dynamically parsing a digital content description of a named digital object, creating and maintaining fragment identities to facilitate caching (Col. 2, lines 36-39) that can be executed on a computing node (a Web server or Proxy server) (Col. 3, lines 61-67; compare to Claim 28, "***A computer software product for use in a computer system that executes program steps recorded in a computer-readable media to perform a method for enabling storing of data pages at a proxy comprising ...***"). Schloss also teaches software for execution on a computer or other processor-based device. The software

may be embodied on a magnetic, electrical, optical, or other persistent program and/or data storage device, including but not limited to: magnetic disks, DASD, bubble memory; tape; optical disks such as CD-ROMs; and other persistent (also called non-volatile) storage devices such as core, ROM, PROM, flash memory, or battery backed RAM (Col. 10, lines 14-22; compare to Claim 28, “... **(a) a recordable media; (b) a program of computer-readable instructions executable by the computer to perform method steps comprising...**”). Schloss also teaches in Fig. 7 an object request handler. In step (705), it is first checked whether the requested object is cached in the object cache maintained by the computing node (Col. 6, lines 31-34; compare to Claim 28, “... **(i) receiving a data page**”). Schloss also teaches a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a plurality of entries (507), where each table entry (507) points to a fragment description list (510) (only one shown for ease of description). The list (510) includes one or more description elements (520 and 525). Each fragment that maps to a given entry in the fragment description table (510) has a unique description element (520) on the fragment description list (510) of the entry. The description element includes several fields: Nlink (530); Fname (535); and Fdescription (540). The Fname (535) is the persistent name of the fragment. This name is given by the persistent name creator routine (with details depicted in Fig. 10). The Fdescription (540) is the fragment description. The Nlink (530) points to the next description element (525) that maps to the same fragment description table entry (507) (Col. 5, lines 66-67; Col. 6, lines 1-15; compare to Claim 28, “... **(ii) receiving page dependency data that**”).

contains one or more dependencies such that each dependency indicates an underlying data source which the said data page is dependent on"). Schloss also teaches that in step (720), the computing node waits for the object requested. In step (725), after receiving the object, the object parser (with details described with reference to Fig. 8) is invoked to analyze the object description and create fragments. In step (730), the object description, which may have been modified by the object parser, is sent back to the requester. In step (735), the object cache manager is invoked to determine whether the object description (which may have been modified by the object parser) should be cached in the object cache (Col. 6, lines 37-46; compare with Claim 28, ***"... (iii) storing said data page; (iv) storing said page dependency data"***).

Schloss fails to specifically teach (a) *receiving an event*; (b) *determining if said event changes one of the said page dependency data associated with said data page*; and (c) *updating the cache by refreshing or deleting said data page*. However, Arlitt teaches that the cache manager (74) controls the eviction of cached objects in the cache (72). Each storage area of the cache (72) is refreshed (i.e., evicting cached objects stored in the cache) in accordance with one of a number of replacement strategies or policies (Col. 5, lines 35-39; compare to Claim 28, ***"... (v) receiving an event; (vi) determining if said event changes said one of the page dependency data associated with said data page; (vii) updating the cache by refreshing or deleting said data page"***). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Arlitt providing the benefit of managing a proxy or web server cache.

In regard to independent Claim 29, Schloss teaches proxy servers (30 . . . 33) or Web content servers (40 . . . 43). The proxy servers and Web servers can provide caching of frequently accessed Web objects to improve client access time (Col. 3, lines 61-67; compare to Claim 29, ***“A proxy server system that provides stored data files without requesting data files from the origin web server, ...”***). Schloss also teaches an Internet environment containing a client (60 ... 63) connected through a network (25) to access proxy servers (30 ... 33) or web content servers (40 ... 43) (Col. 3, lines 61-65; Fig. 1; compare with Claim 29, ***“... (a) a central processing unit that can establish communication with a user computer”***). Schloss also teaches that the computing node can include: a CPU (250); a scratch pad or main memory (245) such as RAM; and persistent storage devices (260) such as direct access storage devices (DASD). The memory (245) stores the server logic 240 (with details depicted in Fig. 6) preferably embodied as computer executable code that may be loaded from DASD (260) into memory (245) for execution by CPU (250) (Col. 4, lines 1-13; compare to Claim 29, ***“... (b) a storage device; (c) a processor connected to the storage device wherein the storage device stores: (i) at least one program component for controlling the processor; and (d) the processor is operative with said program component to ...”***). Schloss also teaches in Fig. 7 an object request handler. In step (705), it is first checked whether the requested object is cached in the object cache maintained by the computing node (Col. 6, lines 31-34; compare to Claim 29, ***“... (i) receive a data page”***). Schloss also teaches a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a

plurality of entries (507), where each table entry (507) points to a fragment description list (510) (only one shown for ease of description). The list (510) includes one or more description elements (520 and 525). Each fragment that maps to a given entry in the fragment description table (510) has a unique description element (520) on the fragment description list (510) of the entry. The description element includes several fields: Nlink (530); Fname (535); and Fdescription (540). The Fname (535) is the persistent name of the fragment. This name is given by the persistent name creator routine (with details depicted in Fig. 10). The Fdescription (540) is the fragment description. The Nlink (530) points to the next description element (525) that maps to the same fragment description table entry (507) (Col. 5, lines 66-67; Col. 6, lines 1-15; compare to Claim 29, “... **(ii) receive page dependency data that contains one or more dependencies such that each dependency indicates an underlying data source which the said data page is dependent on**”). Schloss also teaches that in step (720), the computing node waits for the object requested. In step (725), after receiving the object, the object parser (with details described with reference to Fig. 8) is invoked to analyze the object description and create fragments. In step (730), the object description, which may have been modified by the object parser, is sent back to the requester. In step (735), the object cache manager is invoked to determine whether the object description (which may have been modified by the object parser) should be cached in the object cache (Col. 6, lines 37-46; compare with Claim 29, “... **(iii) store said data page; (iv) store said page dependency data**”). Schloss fails to specifically teach (a) receiving an event; (b) determining if said event changes one of the said page dependency data associated

with said data page; and (c) updating the cache by refreshing or deleting said data page. However, Arlitt teaches that the cache manager (74) controls the eviction of cached objects in the cache (72). Each storage area of the cache (72) is refreshed (i.e., evicting cached objects stored in the cache) in accordance with one of a number of replacement strategies or policies (Col. 5, lines 35-39; compare to Claim 29, “... **(v) receive an event; (vi) determine if said event changes said one of the page dependency data associated with said data page; and (vii) update the cache by refreshing or deleting said data page**”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Arlitt providing the benefit of detecting events that would have triggered a refreshing of a cache.

Claims 14-20, and 26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schloss in view of Arlitt and in further view of Batchelder.

In regard to dependent Claim 14, Schloss fails to specifically teach that *said event came from a Request-based dependency generator*. However, Batchelder teaches a URL Parser (303), which breaks the URL into different parts. The parsed URL is passed to the cache control unit (311) (Col. 7, lines 13-15; compare to Claim 14, “... **said event came from a Request-Based event generator**”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Arlitt, and Batchelder providing the benefit of having provoked a change in the status of data pages on a server.

In regard to dependent Claim 15, Schloss fails to specifically teach that *said Request-Based dependency generator uses a configuration file*. However, Batchelder teaches that Fig. 5 depicts an example of a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a plurality of entries (507), where each table entry (507) points to a fragment description list (510) (only one shown for ease of description). The list (510) includes one or more description elements (520 and 525) (Col. 5, lines 66-67; Col. 6, lines 1-5; compare to Claim 15, “... ***said Request-Based event generator uses a configuration file***”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Arlitt, and Batchelder providing the benefit of having provoked a change in the status of data pages on a server.

In regard to dependent Claim 16, Schloss fails to specifically teach *said Request-Based dependency generator uses a URL request of the said data page*. However, Batchelder teaches a URL Parser (303) that breaks the URL into different parts. Specifically, commands to perform functions are added to the URL (after a question mark separator) (Col. 7, lines 12-27; compare to Claim 16, “... ***said Request-Based event generator uses a URL request of the said data page***”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Arlitt, and Batchelder providing the benefit of having a means to have controlled the actions of a proxy or web server.

In regard to dependent Claim 17, Schloss fails to specifically teach *said Request-Based dependency generator uses a URL request of the said data page*. However,

Batchelder teaches a URL Parser (303) that breaks the URL into different parts. Specifically, commands to perform functions are added to the URL (after a question mark separator) (Col. 7, lines 12-27; compare to Claim 17, “... **said URL request is parsed to obtain parameters**”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Arlitt, and Batchelder providing the benefit of having a means to have controlled the actions of a proxy or web server.

In regard to dependent Claim 18, Schloss fails to specifically teach *said URL request includes request header information*. However, Batchelder teaches sending server-specific commands requesting actions to be taken (Col. 7, lines 12-34; compare to Claim 18, “... **said URL request includes request header information**”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Arlitt, and Batchelder providing the benefit of having a means to have controlled the actions of a proxy or web server.

In regard to dependent Claim 19, Schloss fails to specifically teach *said URL request is parsed to obtain parameters*. However, Batchelder teaches a URL Parser (303) that breaks the URL into different parts. Specifically, commands to perform functions are added to the URL (after a question mark separator) (Col. 7, lines 12-27; compare to Claim 19, “... **said URL request is parsed to obtain parameters**”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Arlitt, and Batchelder providing the benefit of having commanded a server to execute a given action, such as opening a database.

In regard to dependent Claim 20, Schloss fails to specifically teach that *said event came from a script-based dependency generator*. However, Batchelder teaches that once the web page response is built by the response builder (307), it is passed to the HTML unit (305) for conversion to HTML. This HTML response is then passed to the HTTP server (206) for serving to the requesting user (Col. 7, lines 59-62; compare to Claim 20, “... ***said event came from a script-based event generator***”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Arlitt, and Batchelder providing the benefit to the user of not having had to remember a complicated URL by having it hard-coded.

In regard to dependent Claim 26, Schloss fails to teach that *said event generated by Request-Based event generator and sent to a change event evaluator*. However, Batchelder teaches that for those URLs having commands requesting a possibly-cached response (i.e., ?OpenDatabase, ?OpenView, ?OpenDocument, ?OpenForm, and ?ReadForm), the cache control (311) examines the request against previously cached responses to determine whether any of the previously cached responses is appropriate for the request. It does this by comparing the parsed URL against the URLs of the previously cached responses in the cache (304). If there is not an exact match or if the URL doesn't have "cacheable" commands (e.g., ?EditDocument), the parsed URL is passed to the response builder (307) (Col. 7, lines 34-43; compare to Claim 26, “... ***said event generated by Request-Based event generator and sent to a change event evaluator***”). It would have been obvious to one of ordinary skill in the art at the

time of invention to combine the teachings of Schloss, Arlitt, and Batchelder providing the benefit of having sent commands to the server.

Claims 24-25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schloss in view of Arlitt and in further view of Bourne et al. (hereinafter Bourne, U.S. Patent No. 6,584,548).

In regard to dependent Claim 24, Schloss fails to specifically teach that *said determination if said event changes one of the page dependency data associated with said data page is done by a change event evaluator*. However, Bourne teaches a cache coordinator, wherein the cache coordinator invalidates one or more cache entries in response to a signal (Col. 2, lines 49-50; compare to Claim 24, “... ***said determination if said event changes one of the page dependency data associated with said data page is done by a change event evaluator***”). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Arlitt, and Bourne providing the benefit of having determined when a change had taken place.

In regard to dependent Claim 25, Schloss fails to teach that *said determination by the change event evaluator is done by matching said page dependency data with said event*. However, Bourne teaches an ID-based invalidation process, wherein a cache entry is associated with an ID that uniquely identifies the cache entry and can optionally be associated with one or more data ids that represent the underlying data contained in the cache entry, and the ID-based invalidation process sends a signal to

Art Unit: 2176

the cache coordinator to invalidate all cache entries that either have that cache entry ID or have been associated with a data ID when the data that the ID represents changes (Col. 2, lines 51-58; compare to Claim 25, “... ***said determination by the change event evaluator is done by matching said page dependency data with said event***”.

It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Arlitt, and Bourne providing the benefit of having updated a cache.


Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to James H Blackwell whose telephone number is 703-305-0940. The examiner can normally be reached on Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Joseph H Feild can be reached on 703-305-9792. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

James H. Blackwell
04/09/04


JOSEPH FEILD
SUPERVISORY PATENT EXAMINER